

Bayesian Algorithmic Mechanism Design

Jason D. Hartline

Brendan Lucier

September 25, 2009

Abstract

The principal problem in algorithmic mechanism design is in merging the incentive constraints imposed by selfish behavior with the algorithmic constraints imposed by computational intractability. This field is motivated by the observation that the preeminent approach for designing incentive compatible mechanisms, namely that of Vickrey, Clarke, and Groves; and the central approach for circumventing computational obstacles, that of approximation algorithms, are fundamentally incompatible: natural applications of the VCG approach to an approximation algorithm fails to yield an incentive compatible mechanism. We consider relaxing the desideratum of (ex post) incentive compatibility (IC) to Bayesian incentive compatibility (BIC), where truth-telling is a Bayes-Nash equilibrium (the standard notion of incentive compatibility in economics). For welfare maximization in single-parameter agent settings, we give a general black-box reduction that turns any approximation algorithm into a Bayesian incentive compatible mechanism with essentially the same¹ approximation factor. In other words, for a large relevant class of problems and slight relaxation of the standard (in computer science) solution concept, we completely solve the central problem in algorithm mechanism design.

¹More specifically, we obtain an FPTAS: for any ϵ we take a β -approximation algorithm and give a BIC $(1 + \epsilon)\beta$ -approximation mechanism. This $(1 + \epsilon)$ factor arises from statistical methods that seem necessary for a black-box reduction.

1 Introduction

Can any approximation algorithm be converted into an approximation mechanism for selfish agents? This question is framed by a fundamental incompatibility between the standard economic approach for the design of mechanisms for selfish agents (the *Vickrey-Clarke-Groves* (VCG) mechanism) and the standard algorithmic approach for circumventing computational intractability (approximation algorithms). The conclusion from this incompatibility, driving much of the field of algorithmic mechanism design, is that incentive and algorithmic constraints must be dealt with simultaneously. For a large, important class of problems, we arrive at the opposite conclusion: *there is a general approximation-preserving reduction from mechanism design to algorithm design!*

The goal of mechanism design is to construct the rules for a system of agents so that in the equilibrium of selfish agent behavior a desired objective is obtained. For settings of incomplete information the standard game theoretic equilibrium concept is *Bayes-Nash equilibrium* (BNE), which is defined by mutual best response when the *prior distribution* of agent payoffs is commonly known. The *revelation principle* [12] suggests that when looking for mechanisms with desirable Bayes-Nash equilibria, one must look no further than those with truth-telling as a Bayes-Nash equilibrium, also known as *Bayesian incentive compatible* (BIC) mechanisms. Almost all of the computer science literature has focused on the BIC subclass of *ex post incentive compatible* (IC) mechanisms where truth-telling is a *dominant strategy*. While IC is aesthetically appealing because it is compatible with worst-case-style results, it is not generally without loss!

Bayesian mechanism design is very well understood in single-parameter agent settings, such as (for example) where each agent has a single private value for receiving a service (see, e.g., [12]). For the single parameter setting it is known that a mechanism is BIC if and only if (a) the probability an agent is served (a.k.a. the *allocation rule*) is monotone non-decreasing in the agent's value for service, and (b) the expected payment (a.k.a. the *payment rule*) is identified in a particular way from the allocation rule [12]. Probabilities and expectations above are taken with respect to both the distribution of agent values and possible randomization in the mechanism.

The main challenge in reducing BIC (or IC) mechanism design to algorithm design is that approximation algorithms do not generally have monotone allocation rules. Our reduction shows that in a Bayesian setting we can convert any non-monotone allocation rule into a monotone one without compromising its performance. The main technical observation that enables this reduction is that, in a Bayesian setting, we can focus on a single agent for whom the allocation rule is not monotone, apply a transformation that fixes this non-monotonicity (and weakly improves our objective), and *no other agents are affected (in a Bayesian sense)*. Thus, we can apply the transformation independently to each agent.

Our reduction is as follows:

1. For each agent, identify intervals in which the agent's allocation rule is non-monotone. (This is a property of the distribution and can be done prior to considering any agent bids.)
2. For each agent, if their bid falls in an (above identified) interval, redraw the agent's bid from the prior distribution conditioned on being within the interval.
3. Run the approximation algorithm on the resulting bids and output its solution.

Notice that under the assumption that the original values are drawn according to the common prior, the redrawing of values does not alter this prior.

Two items must be clarified. First, there are many ways one might attempt to monotinize a non-monotone function (i.e., choose intervals in step 1 of the reduction), and most of them are incompatible with mechanism design. To address this issue we develop a monotizing technique for allocation rules, adapted from the a standard *ironing procedure* from the field of Bayesian optimal mechanism design (introduced by Myerson [12] for monotizing *virtual valuation functions*, a fundamental construct for the objective of profit maximization). Second, we must also determine payments for our monotized allocation rule. For this, a general approach of Archer et al. [2] suffices.

Selecting intervals of non-monotonicity as described above requires knowing the allocation rule precisely. If our algorithm is only given as a black box, it is unlikely we will have this precision. Therefore, we describe a statistical sampling approach that discretizes the valuation space and estimates the allocation rule in this discrete space to a high level of accuracy. Because this approach may fail to notice and fix a non-monotonicity, to achieve a truly BIC mechanism we will have to take a convex combination of our mechanism with one that is blatantly monotone (but does not approximate our objective). Unsurprisingly, in polynomial time we can achieve arbitrarily precise estimates which means our loss from this convex combination can be made arbitrarily small (resulting in a fully polynomial time approximation scheme (FPTAS)). Alternatively, we could ignore these small non-monotonicities and the resulting mechanism will have truth-telling as an ϵ -Bayes-Nash equilibrium where ϵ can be made arbitrarily small.

The Bayesian approach to mechanism design is relevant for two reasons. The first, as described above, is that it is the standard solution concept in settings of incomplete information. The second is that for many relevant settings the worst-case approximation factor of the best algorithm is so bad that it is completely unrealistic to believe that a designer would be happy with such an approximation.² For instance, for single-minded combinatorial auctions of m items, the optimal worst-case approximation factor (under standard complexity theoretic assumptions) is \sqrt{m} [11]. However, with relevant structure on the desired bundles and a distribution over valuations, it is possible to do much better. This motivates the consideration of the algorithmic problem of Bayesian (a.k.a. stochastic) approximation. Given a distribution over instances, the *Bayesian approximation factor* of the algorithm is the ratio of the expected welfare of the optimal allocation to the expected welfare of the algorithm. Notice that the class of worst-case β -approximations is a subclass of the the class of Bayesian β -approximations. All of our results apply to this more general class of Bayesian β -approximations algorithms.

For Internet settings for mechanism design such as online auctions (eBay), advertising auctions (Google, Yahoo!, MSN), file sharing (BitTorrent), routing (TCP/IP), scheduling (SETI@home), video streaming (YouTube), etc. it is quite reasonable to believe that detailed statistics about distributions of user demand are available to the designer. It would be impractical to ignore this information in designing mechanisms and from a lower-bounding point of view unrealistic not to assume the distribution is known. The BIC equilibrium notion is the relevant one for these settings. Note that it is not the case that agents in a BIC mechanism must themselves know the common prior distribution, as long as they trust that the mechanism designer does. Thus, we posit that that BIC and Bayesian approximation are the appropriate notions for a theory of mechanism design that is relevant to Internet systems.

Our results apply generally to single-parameter agent settings where the designer's objective is to maximize the social welfare (e.g., single-minded combinatorial auctions). In the most general

²This is the basis for the economics community's dissatisfaction with worst-case approximation.

form, such an algorithmic problem can be written as finding an allocation $\mathbf{x} = (x_1, \dots, x_n)$ to maximize $\sum_i v_i x_i - c(\mathbf{x})$ for agent valuations $\mathbf{v} = (v_1, \dots, v_n)$ and cost function $c(\cdot)$. For instance, the *multicast auction* problem of Feigenbaum et al. [10] is the special case where the $c(\mathbf{x})$ is the sum of the costs of tree edges necessary to connect all agents served by \mathbf{x} to the root (generally, the Steiner tree problem). A special and relevant case occurs when costs are zero for \mathbf{x} in some feasible set system \mathcal{X} and all other allocations are infeasible (i.e. $c(\mathbf{x}) = 0$ if $\mathbf{x} \in \mathcal{X}$ and ∞ otherwise). For the single-item auction, \mathcal{X} is the collection of all sets of cardinality at most one; and for single-minded combinatorial auctions, \mathcal{X} contains all sets of agents with non-intersecting desired bundles. Our most general result does not need any restrictions on the cost function or the set system. In particular, costs can be arbitrarily non-monotone or the set system non-downward-closed (e.g. public good or scheduling problems).

For the most studied single-parameter mechanism design problems, the performance of the best approximation mechanism matches the best approximation algorithm (e.g., single-minded combinatorial auctions [11] and related machine scheduling [9]). None-the-less, our approach gives the best known approximation mechanism for many problems, such as auctions under various graph constraints [1] and auctions of convex bundles [4].

Our result that there is no gap between algorithmic approximation and approximation by BIC mechanisms essentially solves the fundamental problem single-parameter algorithmic mechanism design. The remaining (theoretical) question of gaps in approximation factors imposed by incentive constraints is thus focused on whether BIC is more powerful than IC for social welfare maximization. For other non-welfare-maximization objectives (e.g. makespan) the question of a general reduction remains open.

Related Work Lehmann et al. [11] introduced the problem of computing the single-minded combinatorial auction and give a mechanism that matches the best algorithmic approximation factor. Archer et al. [2] considered the setting where there are many (at least logarithmic) copies of each item and gave a $(1 + \epsilon)$ -approximation mechanism. Archer and Tardos [3] gave (single-parameter) related machine scheduling mechanism that approximates the makespan. Dhangwatnotai et al. [9] gave an approximation mechanism for related machine scheduling that approximates makespan and matches the algorithmic lower bound. All of the above results are for ex post incentive compatible mechanisms.

There has been a large literature on multi-parameter combinatorial auctions and approximation, but this is only tangentially related to our work so we do not cite it exhaustively.

Babaioff et al. [5] looks at the equilibrium notion of *algorithmic implementation in undominated strategies* and gives a general technique for turning an β -algorithm into a $\beta(\log v_{max})$ -approximation mechanism. This solution concept requires that no agent plays a strategy that is dominated by an easy to find strategy. Their approach applies to single-valued combinatorial auctions and does not require the mechanism to know which bundles each agent desires.

Christodoulou et al. [8] looks at Bayes-Nash equilibrium in simultaneous Vickrey auctions in a combinatorial setting and shows that these give a 2-approximation when agents' valuations are submodular.

There are many papers for profit maximization that consider Bayesian design settings. These papers do not tend to consider computational constraints and for many of these (non-computational) settings the restriction to ex post incentive compatibility is without loss. One example where ex post incentive compatibility is with loss is when the profit maximizing seller has a strict no-deficit

constraint [7].

To our knowledge no prior work looks at computational questions related to approximation of social welfare by a Bayesian incentive compatible mechanism.

2 Model and Definitions

Algorithms. We consider algorithms for binary single-parameter agent settings. An algorithm in such a setting must select a set of agents to serve. This *allocation* is denoted by $\mathbf{x} = (x_1, \dots, x_n)$ where x_i is an indicator for whether or not agent i is served. Agent i has *valuation* v_i for being served. The vector $\mathbf{v} = (v_1, \dots, v_n)$ of valuations is the *valuation profile*. The seller may have some cost function $c(\cdot)$ over allocations that represents the cost of serving the allocated set. A special case are zero-infinity cost functions where the set of allocations with zero cost are *feasible* and those with infinite cost are *infeasible*.

The objective we consider is *social welfare maximization*, i.e., to find the allocation \mathbf{x} to maximize $\sum_i v_i x_i - c(\mathbf{x})$. We refer to this as the *general costs setting* and it includes, for example, Steiner tree problems. We refer to the special case where the cost function is zero or infinite as the *general feasibility setting*. General feasibility problems include scheduling and public good problems. An important subclass are *downward-closed (feasibility) settings* where any subset of a feasible set is feasible. Downward closed settings include single-minded combinatorial auctions and knapsack problems. For any setting, $\text{OPT}(\mathbf{v})$ will denote the maximum social welfare.

An algorithm \mathcal{A} is simply an *allocation rule* that maps valuation profiles to allocations. The allocation rule for \mathcal{A} we will denote by $\mathbf{x}(\mathbf{v})$. The social welfare of an algorithm is $\mathcal{A}(\mathbf{v}) = \sum_i v_i x_i(\mathbf{v}) - c(\mathbf{x}(\mathbf{v}))$. We allow \mathcal{A} to be randomized in which case $x_i(\mathbf{v})$ is a random variable; $\mathcal{A}(\mathbf{v})$ denotes the expected welfare of the algorithm for valuation profile \mathbf{v} .

We will be considering these algorithmic problems in a Bayesian (a.k.a. stochastic) setting where the valuations of the agents are drawn from a product distribution $\mathbf{F} = F_1 \times \dots \times F_n$. Agent i 's *cumulative distribution* and *density* functions are denoted F_i and f_i , respectively. The distributions are assumed to be *common knowledge* to the agents and designer. Some of our bounds will be based on v_{\max} , an upper bound on any agent's valuation, and $\mu_{\max} = \max_i \mathbf{E}[v_i]$, the maximum expected valuation of any agent.

The pair $(c(\cdot), \mathbf{F})$ defines a setting for single-parameter algorithm design which we will take as implicit. For this setting, the optimal expected welfare is $\text{OPT} = \mathbf{E}_{\mathbf{v} \sim \mathbf{F}}[\text{OPT}(\mathbf{v})]$ and the algorithm's expected welfare is $\mathcal{A} = \mathbf{E}_{\mathbf{v} \sim \mathbf{F}}[\mathcal{A}(\mathbf{v})]$. An algorithm is a *worst-case β -approximation* if for all \mathbf{v} , $\mathcal{A}(\mathbf{v}) \geq \text{OPT}(\mathbf{v})/\beta$. An algorithm is a *Bayesian β -approximation* if $\mathcal{A} \geq \text{OPT}/\beta$.

Mechanisms. A mechanism \mathcal{M} consists of an *allocation rule* and a *payment rule*. We denote by $\mathbf{x}(\mathbf{v})$ and $\mathbf{p}(\mathbf{v})$ the allocation and payment rule of an implicit mechanism \mathcal{M} . We assume the agents are *risk neutral* and individually desire to maximize their expected *utilities*. Agent i 's utility for allocation \mathbf{x} and payments \mathbf{p} is $v_i x_i - p_i$. We will consider single-round, sealed-bid mechanisms where agents simultaneously bid and the mechanism then computes the allocation and payments.

Our goal is a mechanism that has good social welfare in equilibrium. The standard economic notion of equilibrium for games of incomplete information is *Bayes-Nash equilibrium* (BNE). The revelation principle says that any equilibrium that is implementable in BNE is implementable with truthtelling as the BNE strategies of the agents.³ Meaning: an agent that believes that the other

³The revelation principle holds even in computational settings; any BNE for which the agent strategies and the

agents are reporting their values truthfully as given by the distribution has a best response of also reporting truthfully. A mechanism with truthtelling as a BNE is *Bayesian incentive compatible* (BIC).⁴

It will be useful to consider agent i 's expected payment and probability of allocation conditioned on their value. Denote $p_i(v_i) = \mathbf{E}_{\mathbf{v}, \mathcal{M}}[p_i(\mathbf{v}) \mid v_i]$ and $x_i(v_i) = \mathbf{E}_{\mathbf{v}, \mathcal{M}}[x_i(\mathbf{v}) \mid v_i]$. The following theorem characterizes BIC mechanisms.

Theorem 2.1 [12] *A mechanism is BIC if and only if for all agents i :*

- $x_i(v_i)$ is monotone non-decreasing, and
- $p_i(v_i) = v_i x_i(v_i) - \int_0^{v_i} x_i(z) dz + p_i(0)$.

Usually, $p_i(0)$ is assumed to be zero.

This motivates the following definition:

Definition 2.1 *An allocation rule $\mathbf{x}(\cdot)$ is monotone for distribution \mathbf{F} if $x_i(v_i)$ is monotone non-decreasing for all i . An algorithm is monotone if its allocation rule is monotone.*

From Theorem 2.1, BIC and monotone are equivalent and we will use them interchangeably for both algorithms and mechanisms, though we will prefer “BIC” when the focus is incentive properties and “monotone” when the focus is algorithmic properties.

Computation. Our main task in demonstrating that the approximation complexity of algorithms and BIC mechanisms is the same by giving an approximation-preserving reduction from the BIC mechanism design problem to the algorithm design problem. In other words, we use the algorithm's allocation rule to compute the mechanism's allocation and payment rules. As we are in a Bayesian setting this computation will also need access to the distribution. We consider two models of computation: an *ideal model* and an *oracle model*.

In the ideal model, we will assume we have explicit access to the functional form of the distribution and allocation rule and we will assume we can perform calculus on these functions. While this model is not realistic, we present it for the sake of clearly explaining the economic theory that drives our results. In the oracle (a.k.a. black-box) model we will assume we can query the algorithm on any input and that we can sample from the distribution on any subinterval of its support. Our philosophy is that the ideal model is predictive of what is implementable in polynomial time and we verify this philosophy by instantiating approximately the same reduction under the oracle model.

We give our algorithm runtimes in the oracle model in terms of the number of calls to the algorithm oracle. This is justified as each such call requires at most a linear number of calls to the distribution oracle and no significant additional computation is needed.

mechanism can be computed in polynomial time can be converted into a polynomial time BIC mechanism.

⁴Much of the computer science literature on mechanism design focuses on dominant strategy equilibrium (DSE) and *ex post incentive compatibility* (IC). This is not without loss in many settings and therefore should be considered with care when addressing computational questions in mechanism design. Whether it is without loss in computational settings is an open question.

3 Reduction: the Ideal Model

Theorem 3.1 *In the ideal model and general cost settings, a BIC Bayesian β -approximation $\bar{\mathcal{A}}$ can be computed from any Bayesian β -approximation algorithm, \mathcal{A} .*

Suppose that we are given an algorithm \mathcal{A} that is monotone for the distribution \mathbf{F} . Then \mathcal{A} is already BIC and specifies the allocation rule, so we must only compute the payment rule. In our ideal model this is trivial given the formula from Theorem 2.1.

Now suppose we have a non-monotone Bayesian c -approximation algorithm \mathcal{A} with allocation rule $\mathbf{x}(\cdot)$. We would like to use \mathcal{A} to construct a monotone algorithm $\bar{\mathcal{A}}$ from which we can obtain a BIC mechanism by simply computing the payment rule as above. We must make sure that in doing so we do not reduce the algorithm’s expected welfare. The key property of our approach which makes it tractable is that we monotinize each agent’s allocation rule independently without changing (in a Bayesian sense) the allocation rule any other agent faces. This property is also important for the approximation factor as \mathcal{A} is guaranteed to be a Bayesian c -approximation only for the given distribution \mathbf{F} , and may not be a good approximation for some other distribution.

In summary, the desiderata for monotonizing agent i are:

- D1. monotone $\bar{x}_i(v_i)$,
- D2. (weakly) improved welfare $\mathbf{E}_{v_i}[v_i \bar{x}_i(v_i)] \geq \mathbf{E}_{v_i}[v_i x_i(v_i)]$, and
- D3. other agents unaffected.

Notice that if we satisfy the last condition we can apply the process simultaneously to all agents.

3.1 Ironing via Resampling

There is a history of fixing non-monotonicities in Bayesian mechanism design. Notably, Myerson invented a technique of *ironing* which relies on the fact that if an allocation rule is constant over some interval then any agent within that interval is effectively equivalent to a canonical “average” agent from that interval. Myerson applied this theory to iron *virtual valuation functions* which are a crucial construct in Bayesian profit maximization [12]. We will apply this theory directly to the allocation rule.

Before we describe our ironing procedure in full, let us develop some intuition. Suppose allocation rule $x_i(\cdot)$ of \mathcal{A} is non-monotone for agent i . A simple approach to flattening non-monotonicities is to choose some interval $[a, b]$ on which $x_i(\cdot)$ is non-monotone, and treat agent i identically whenever $v_i \in [a, b]$. For example, whenever $v_i \in [a, b]$ we could choose to pretend that v_i is actually some other fixed value v' (e.g. $v' = a$), and pass this “pretend” value v' to the algorithm. Unfortunately, if we take this naïve approach, we would have changed the distribution of agent i ’s input to the algorithm (in particular, the probability of value v' would be increased) and violated D3. In order to maintain D3 we make a minor modification: instead of picking a fixed v' , we will draw v' from F_i restricted to the interval $[a, b]$. Thus, we are replacing $v_i \in [a, b]$ with v' drawn from the same distribution. Other agents cannot tell the difference – this operation does not change the distribution of agent i ’s input! Moreover, agent i will indeed be treated identically whenever $v_i \in [a, b]$: the new probability of allocation will be precisely the distribution weighted average of $x_i(\cdot)$ over interval $[a, b]$.

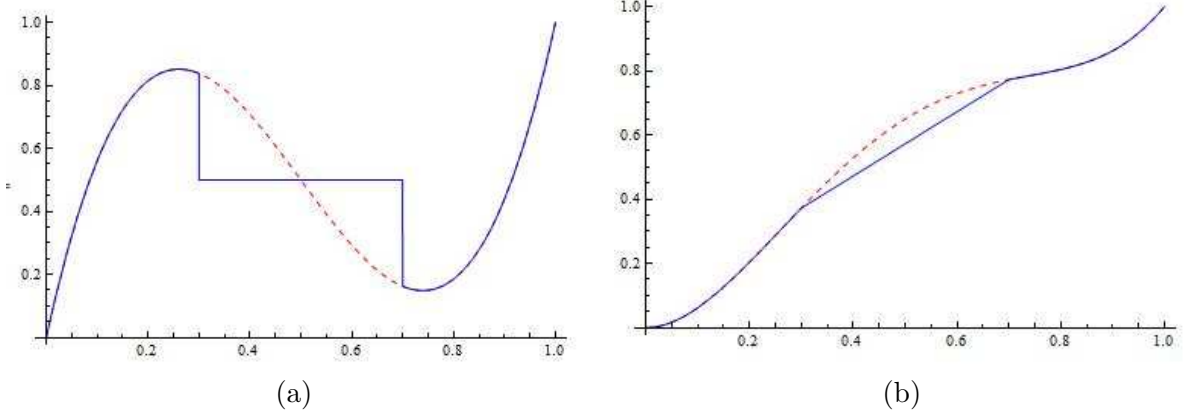


Figure 1: (a) A non-monotone ironing x_i' (solid) of curve x_i (dashed). (b) The corresponding integral curves G (dashed) and G' (solid) in probability space.

Let $x_i'(v_i)$ represent the allocation rule we obtain from the following procedure (when $\mathbf{v}_{-i} \sim \mathbf{F}_{-i}$):

1. if $v_i \in [a, b]$, redraw $v' \sim F_i[a, b]$,
2. else, set $v' = v_i$.
3. run $\mathcal{A}(v', \mathbf{v}_{-i})$.

We say that $x_i'(\cdot)$ is the curve $x_i(\cdot)$ *ironed on interval* $[a, b]$. We note that $x_i'(v_i) = x_i(v_i)$ for $v_i \notin [a, b]$ and $x_i'(v_i) = \mathbf{E}_{v' \sim F_i}[x_i(v') \mid v' \in [a, b]]$ otherwise. This is illustrated in Figure 1(a). Note that we can easily iron along multiple disjoint intervals, redrawing v' from whichever interval contains v_i (if any).

Figure 1(a) demonstrates that an ironed allocation rule is not necessarily monotone. We now explore a method for choosing intervals on which to iron in order to obtain monotonicity. It will be instructive to consider the allocation rule in probability space instead of valuation space and the *cumulative allocation rule* (also in probability space).

1. Let $g(q) = x_i(F_i^{-1}(q))$ be the allocation rule in probability space.
2. Let $G(q) = \int_0^q g(z)dz$ be the cumulative allocation rule.

We make two important observations: x_i is monotone if and only if g is monotone, and g is monotone if and only if G is convex. Let x_i' be x_i ironed on interval $[a, b]$, and consider the corresponding curves g' and G' . Then G' and G are equal, except on $[a, b]$ where G' is a straight line connecting $(a, G(a))$ to $(b, G(b))$. See Figure 1(b). We can therefore view our interval selection problem as the problem of replacing portions of curve G with straight line segments, so that the resulting curve G' will be convex. This is precisely the problem of finding the convex hull of G ! Thus the choice of intervals that monotonicizes x_i is precisely the set of intervals defined by the convex hull of G .

3. Let $\bar{G}(\cdot)$ be the convex hull of $G(\cdot)$.
4. Let $\mathcal{I}_i = \{I_1, I_2, \dots\}$ be the intervals in valuation space where $G(F_i(v_i)) > \bar{G}(F_i(v_i))$

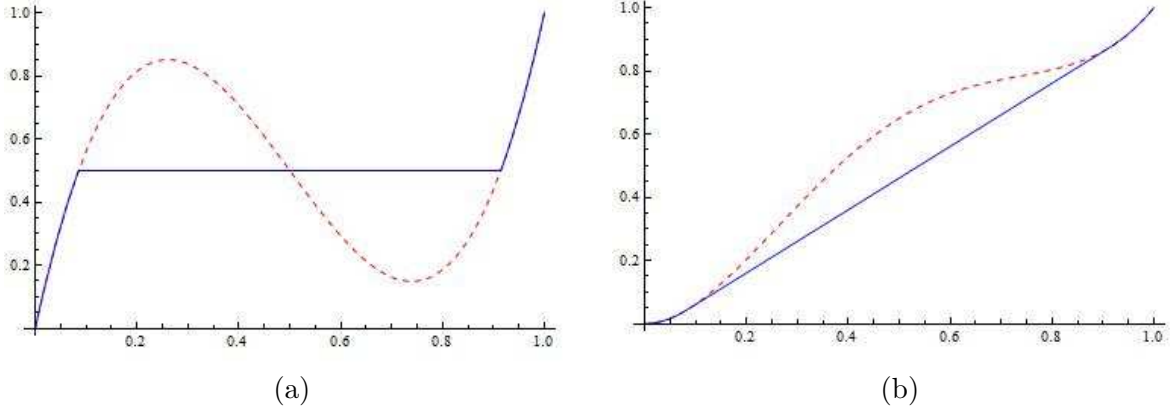


Figure 2: (a) A monotone ironing (solid) of curve x_i (dashed). (b) The corresponding integral curves G (dashed) and \bar{G} (solid) in probability space. Note \bar{G} is the convex hull of G .

We refer to \mathcal{I}_i as the set of monotonizing intervals for x_i . Once we have found \mathcal{I}_i , we simply apply our ironing procedure to the intervals in \mathcal{I}_i . That is, whenever $v_i \in I \in \mathcal{I}_i$, we redraw \bar{v} from the distribution restricted to I and input \bar{v} into \mathcal{A} instead of v_i . The resulting allocation rule will be $\bar{x}_i(v_i) = \bar{g}(F_i(v_i))$, which is monotone since \bar{G} is convex. See Figure 2.

3.2 The Ironed Algorithm

We are now ready to define our ironed algorithm $\bar{\mathcal{A}}$.

Definition 3.1 ($\bar{\mathcal{A}}$) *Given a profile of disjoint interval sets $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$, the ironed algorithm $\bar{\mathcal{A}}_{\mathcal{I}}$ for \mathcal{A} with intervals \mathcal{I} is:*

1. For each agent i , if $v_i \in I \in \mathcal{I}_i$, draw $\bar{v}_i \sim F_i[I]$; else, set $\bar{v}_i = v_i$.
2. Run $\mathcal{A}(\bar{\mathbf{v}})$.

The ironed algorithm $\bar{\mathcal{A}}$ for \mathcal{A} computes the monotonizing intervals \mathcal{I}_i for each agent, as described in the previous section, then runs $\bar{\mathcal{A}}_{\mathcal{I}}(\mathbf{v})$.

Lemma 3.2 $\bar{\mathcal{A}}$ is monotone.

Proof: It is enough to show that each agent has a monotone allocation rule. By construction the allocation rule satisfies $\bar{x}_i(v_i) = \bar{g}(F_i(v_i))$, which is the derivative of a convex function and therefore monotone. \square

Lemma 3.3 If \mathcal{A} is a Bayesian β -approximation then $\bar{\mathcal{A}}$ is a Bayesian β -approximation.

Proof: First notice that the two allocation rules produce the same distribution over allocations and therefore expected costs are identical. We will show, for a single agent i , that $\mathbf{E}[v_i \bar{x}_i(v_i)] \geq \mathbf{E}[v_i x_i(v_i)]$ (i.e., D2), from which linearity of expectation implies the result. In fact, the former stochastically dominates the latter: for any v_i , $\int_{v_i}^{\infty} \bar{x}_i(z) f_i(z) dz \geq \int_{v_i}^{\infty} x_i(z) f_i(z) dz$, since $\bar{G}(q) \leq$

$G(q)$ for all q and thus there is more weight in the expectation on higher values for $\bar{x}_i(\cdot)$ than for $x_i(\cdot)$. \square

Theorem 3.1 follows immediately from Lemmas 3.2 and 3.3. An immediate corollary for worst-case approximation algorithms follows since worst-case approximations are a subclass of Bayesian approximations.

Corollary 3.4 *In the ideal model and general cost settings, a BIC Bayesian β -approximation $\bar{\mathcal{A}}$ can be computed from any worst-case β -approximation algorithm, \mathcal{A} .*

Notes. We make the following notes about our main result (detailed discussion is given in our conclusions).

- The argument fails for non-linear objectives such as makespan. While D2 holds, it will not lead to an overall bound on the expected performance. See Appendix A.1 for an example.
- Our ironing procedure is distinct from Myerson’s. Myerson irons virtual valuations and allocates to maximize ironed virtual value. This is not the same as maximizing virtual value and then ironing the allocation rule where it is non-monotone. See Appendix A.3 for an example.
- If \mathcal{A} is a worst-case c -approximation, $\bar{\mathcal{A}}$ may fail to be a worst-case c -approximation. See Appendix A.2 for an example.

4 Reduction: the Oracle Model

We will use the ironing procedure from the previous section in order to monotinize an algorithm in the oracle model. In the ideal model, we used knowledge of an algorithm’s allocation rule in order to find the monotizing intervals for each agent. In the oracle model our approach will be the same, except that instead of using direct knowledge of the allocation rule we must use sampling to estimate it. This sampling introduces errors which must then be dealt with. Our analysis will proceed in the following steps.

1. We describe a method for computing payments in the oracle model.
2. We describe a method for combining sampling with ironing to obtain a nearly monotone algorithm. In fact, this algorithm will be ϵv_{max} -Bayesian incentive compatible.⁵
3. We show that a convex combination of this nearly monotone algorithm with a blatantly monotone one will give a monotone algorithm. Coupled with the payment rule, this gives a Bayesian incentive compatible mechanism.

All of these steps approximately preserve social welfare. We obtain the following main theorem and corollary. (Recall: v_{max} is an upper bound on any valuation and μ_{max} is the maximum over the agents’ expected valuations.)

⁵This intermediate result may be desirable in itself. If v_{max} is polynomially bounded, we can get an ϵ -BIC mechanism in polynomial time.

Theorem 4.1 *In the oracle model and general cost settings, for any $\epsilon > 0$, a BIC algorithm $\hat{\mathcal{A}}_\epsilon$ can be computed from any Bayesian algorithm \mathcal{A} . Its social welfare satisfies $\hat{\mathcal{A}}_\epsilon \geq \mathcal{A} - \epsilon\mu_{\max}$. The runtime is $\tilde{O}(n^9\epsilon^{-9}\log^5(v_{\max}/\epsilon\mu_{\max}))$. In general feasibility settings, the runtime can be improved to $\tilde{O}(n^5\epsilon^{-5}\log^3(v_{\max}/\epsilon\mu_{\max}))$.*

For the special case of downward-closed set systems for feasibility problems, we can assume that $\mathcal{A} \geq \mu_{\max}$, since the trivial algorithm that simply allocates to the single player with the highest input value attains this value. This implies the following corollary.

Corollary 4.1 *In the oracle model and downward-closed feasibility settings, for any $\epsilon > 0$, a BIC Bayesian $\beta(1+\epsilon)$ -approximation algorithm $\hat{\mathcal{A}}_\epsilon$ can be computed from any Bayesian β -approximation algorithm \mathcal{A} . Its runtime is $\tilde{O}(n^5\epsilon^{-5}\log^3(v_{\max}/\epsilon\mu_{\max}))$.*

4.1 Computing Payments

Suppose that \mathcal{A} has monotone allocation rules. The problem of designing a mechanism to implement \mathcal{A} then reduces to calculating appropriate payments. These payments are completely determined by the allocation rule of \mathcal{A} , but in the oracle model we do not have direct access to the functional form of the allocation rule. Archer et al. [2] solve this problem by computing an unbiased estimator of the desired payment rule using only oracle calls to the algorithm. For completeness we summarize their approach below.

Definition 4.1 (oracle payment procedure) *If \mathcal{A} does not allocate to agent i , then agent i pays 0. Otherwise, we compute the payment of agent i as follows:*

1. Choose v_i' uniformly from $[0, v_i]$
2. Draw $\mathbf{v}'_{-i} \sim \mathbf{F}_{-i}$ and run $\mathcal{A}(v_i', \mathbf{v}'_{-i})$
3. If \mathcal{A} allocated to agent i in the previous step set $X = v_i$, otherwise set $X = 0$.
4. If $X \neq 0$, repeatedly draw $\mathbf{v}'_{-i} \sim \mathbf{F}_{-i}$ and run $\mathcal{A}(v_i, \mathbf{v}'_{-i})$ until the algorithm allocates to player i , and let T be the number of iterations required.
5. Agent i 's payment is $p_i = v_i - TX$.

As was shown by Archer et al., this computation attains the appropriate expected payment.

Claim 4.2 (Archer et al. [2]) *The oracle payment procedure produces payments $p_i(v_i) = v_i x_i(v_i) - \int_0^{v_i} x_i(z) dz$.*

We note that since we execute this procedure for agent i only if he receives an allocation, which occurs with probability $x_i(v_i)$, the expected number of calls to \mathcal{A} for each player is at most

$$x_i(v_i) \left(1 + \frac{1}{x_i(v_i)} \right) \leq 2.$$

Thus, in expectation, all payments can be computed with $2n$ calls to \mathcal{A} .

Notice that any mechanism paired with this payment scheme will be *individually rational* (IR), meaning that a truthtelling agent will never obtain negative utility. This is true even if the allocation rule is not monotone. This follows immediately from the fact that the payment for an agent that declares value v_i is never greater than v_i (indeed, it is defined as v_i minus a non-negative value).

4.2 Sampling and ϵ -Bayesian Incentive Compatibility

We will be estimating the allocation rule of our non-monotone algorithm and attempting to iron it. This will fail to result in an absolutely monotone rule. In this section we show that a nearly monotone rule results in truthtelling as an ϵ -Bayes-Nash equilibrium (ϵ -BNE): the most an agent can gain from a non-truthtelling strategy is an additive ϵ . We call such a mechanism ϵ -Bayesian incentive compatible (ϵ -BIC). While there is no characterization of payment rules for ϵ -BIC mechanisms, the payment rule given by Theorem 2.1 will be sufficient.

Definition 4.2 (ϵ -BIC) *A mechanism is ϵ -Bayesian incentive compatible if truthtelling obtains at least as much utility as any other strategy, up to an additive ϵ , assuming all other agents truthtell. That is, for all i , v_i , and v' ,*

$$v_i x_i(v_i) - p_i(v_i) \geq v_i x(v') - p_i(v') - \epsilon.$$

The main theorem of this section is the following.

Theorem 4.2 *In the oracle model and general cost settings, for any $\epsilon > 0$, an ϵv_{\max} -BIC algorithm \mathcal{A}' can be computed from any Bayesian algorithm \mathcal{A} . Furthermore, $\mathcal{A}' \geq \mathcal{A} - \epsilon \mu_{\max}$. The running time of \mathcal{A}' is at most $\tilde{O}(n^3 \epsilon^{-2} \log_{1+\epsilon}(v_{\max}/\epsilon \mu_{\max}))$.*

As in Corollary 4.1, for the special case of downward-closed set systems, we can assume that $\mathcal{A} \geq \mu_{\max}$ (since any algorithm could simply choose to allocate only to the one player with highest expected value). This implies the following corollary.

Corollary 4.3 *In the oracle model, for any $\epsilon > 0$, an ϵv_{\max} -BIC Bayesian $\beta(1+\epsilon)$ -approximation algorithm \mathcal{A}' can be computed from any Bayesian β -approximation algorithm \mathcal{A} for a downward-closed set system. The running time of \mathcal{A}' is at most $\tilde{O}(n^3 \epsilon^{-2} \log_{1+\epsilon}(v_{\max}/\epsilon \mu_{\max}))$.*

4.2.1 ϵ -closeness

We now formalize a closeness property under which an allocation rule that is close to monotone is ϵ -BIC (for some related ϵ).

Definition 4.3 (ϵ -close) *Allocation rules $x(\cdot)$ and $x'(\cdot)$ are ϵ -close if $|x(v) - x'(v)| < \epsilon$ for all v . Two algorithms or mechanisms are ϵ -close if each agent's allocation rules are ϵ -close.*

Lemma 4.4 *If non-monotone \mathcal{A}' is ϵ -close to a monotone \mathcal{A} , then \mathcal{A}' is $(2\epsilon v_{\max})$ -BIC.*

Proof: Suppose agent i is participating in \mathcal{A}' and has value v_i , but claims to have value v_i' . Assume $v_i > v_i'$; the opposite case is similar. Using the payment rule from Theorem 2.1, agent i 's gain in utility from declaring v_i' is:

$$(v_i x_i'(v_i') - p_i(v_i')) - (v_i x_i'(v_i) - p_i(v_i)) = (v_i - v_i') x_i'(v_i') - \int_{v_i'}^{v_i} x_i'(z) dz. \quad (1)$$

Since $x_i'(\cdot)$ is ϵ -close to a monotone curve, it must be that $x_i'(z) + \epsilon \geq x_i'(v_i') - \epsilon$ for all $z \in [v_i', v_i]$. Thus $\int_{v_i'}^{v_i} x_i'(z) dz \geq (v_i - v_i')(x_i'(v_i') - 2\epsilon)$. This implies that the value in (1) is at most $2\epsilon(v_i - v_i')$, which is at most $2\epsilon v_{\max}$. \square

Now we show that if two algorithms have ϵ -close allocation rules, then they obtain similar welfare. In the statement of Lemma 4.5 we assume that the expected costs of both rules are the

same. This is implied by our techniques: since our mechanism runs the original algorithm on values from the original distribution, the expected cost of our mechanism is equal to the expected cost of the original algorithm.

Lemma 4.5 *If \mathcal{A} and \mathcal{A}' have the same expected costs and are ϵ -close then $\mathcal{A}' \geq \mathcal{A} - n\epsilon\mu_{\max}$.*

Proof: For each agent i , $\mathbf{E}[v_i x_i'(v_i)] \geq \mathbf{E}[v_i(x_i(v_i) - \epsilon)] \geq \mathbf{E}[v_i x_i(v_i)] - \epsilon \mathbf{E}[v_i]$. The result then follows by linearity of expectation. \square

Finally, we note that the property of ϵ -closeness persists when algorithms are ironed on some collection of intervals.

Lemma 4.6 *If algorithms \mathcal{A} and \mathcal{A}' are ϵ -close, then for any collection $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$ of interval sets, ironed algorithms $\bar{\mathcal{A}}_{\mathcal{I}}$ and $\bar{\mathcal{A}}'_{\mathcal{I}}$ are ϵ -close.*

Proof: For any i , let x_i and x_i' be the allocation rules of \mathcal{A} and \mathcal{A}' , respectively. Let \bar{x}_i and \bar{x}_i' be the allocation rules of $\bar{\mathcal{A}}_{\mathcal{I}}$ and $\bar{\mathcal{A}}'_{\mathcal{I}}$. Then for any $I \in \mathcal{I}_i$,

$$|\bar{x}_i(I) - \bar{x}_i'(I)| = |E_v[x(v) \mid v \in I] - E_v[x'(v) \mid v \in I]| = |E_v[x(v) - x'(v) \mid v \in I]| < \epsilon.$$

\square

4.2.2 Discretization

A key step in our reduction will be in discretizing the allocation rules of the algorithm. This is needed for two reasons:

- We need to estimate the allocation rule and so we need to pick a polynomial number of points on it to estimate. (This is the focus of the next step.)
- Our resulting allocation will be almost monotone. By discretizing we will end up with a polynomial number of points in which it is non-monotone and we will know exactly where these points are. (In a subsequent step, a convex combination of the sampled algorithm with a blatantly monotone one will fix the non-monotonicities.)

We say that an algorithm is k -piece piecewise constant if for each i there is a partition of valuation space into at most k intervals such that the allocation rule for agent i is constant on each interval. Our discretization procedure, which converts any algorithm into a piecewise constant algorithm, is the following.

Definition 4.4 ($\dot{\mathcal{A}}_{\epsilon}$) *Given algorithm \mathcal{A} and $\epsilon > 0$, the discretization of \mathcal{A} , $\dot{\mathcal{A}}_{\epsilon}$, is the ironed algorithm $\bar{\mathcal{A}}_{\mathcal{I}}$, where $\mathcal{I} = \{\mathcal{I}_1, \dots, \mathcal{I}_n\}$ is the collection of intervals defined by*

$$\mathcal{I}_i = \{[0, \epsilon\mu_{\max})\} \cup \{[\epsilon\mu_{\max}(1 + \epsilon)^t, \epsilon\mu_{\max}(1 + \epsilon)^{t+1})\}_{0 \leq t \leq \log_{1+\epsilon}(v_{\max}/\epsilon\mu_{\max})}.$$

Lemma 4.7 *For any algorithm \mathcal{A} and $\epsilon > 0$, $\dot{\mathcal{A}}_{\epsilon}$ is $\log_{1+\epsilon}(v_{\max}/\epsilon\mu_{\max})$ -piece piece-wise-constant for all i , and $\dot{\mathcal{A}}_{\epsilon} \geq \mathcal{A} - 2n\epsilon\mu_{\max}$.*

Proof: The allocation curves for $\dot{\mathcal{A}}_{\epsilon}$ are constant on interval $[0, \epsilon\mu_{\max})$ and all intervals of the form $[\epsilon\mu_{\max}(1 + \epsilon)^t, \epsilon\mu_{\max}(1 + \epsilon)^{t+1})$, and there are at most $\log_{1+\epsilon}(v_{\max}/\epsilon\mu_{\max})$ such intervals over the range $[\epsilon'\mu_{\max}, v_{\max}]$. These intervals do, indeed, partition valuation space. Furthermore, $\mathbf{E}_{v_i}[v_i \dot{x}_i(v_i)] \geq (1 - \epsilon)\mathbf{E}_{v_i}[v_i x_i(v_i)] - \epsilon\mu_{\max} \geq \mathbf{E}_{v_i}[v_i x_i(v_i)] - 2\epsilon\mu_{\max}$, as algorithm $\dot{\mathcal{A}}_{\epsilon}$ modifies any input value greater than $\epsilon\mu_{\max}$ by at most a factor of $(1 - \epsilon)$. As the expected costs before and after discretization are the same, the result follows from linearity of expectation. \square

4.2.3 Statistical Estimation

We next describe a sampling procedure for estimating an allocation rule. This procedure will not form an algorithm, but rather generates an estimated allocation curve, which we will denote by $\mathbf{y}(\cdot)$. This estimate behaves like an allocation rule, but is not associated with an actual algorithm. Nevertheless, we can pretend that it is an algorithm and apply Lemma 4.4 and Lemma 4.5 to it. The appropriate way to view this is “if \mathbf{y} was an allocation rule for an algorithm, then ...”

Definition 4.5 (estimated allocation rule) *Given algorithm \mathcal{A} which is k -piece piece-wise-constant and $\epsilon > 0$, an estimated allocation rule for \mathcal{A} is a curve $\mathbf{y}(\cdot)$ found as follows:*

1. *for each agent i and valuation-space piece I_j , draw $4(\epsilon)^{-2} \log(2k/n\epsilon)$ samples from \mathbf{F} conditional on $v_i \in I_j$, and run \mathcal{A} on each of these samples.*
2. *let y_{ij} be the average allocation over the invocations to \mathcal{A} above, for each i and j .*
3. *Define \mathbf{y} by $y_i(v) = y_{ij}$ for all $v \in I_j$*

Lemma 4.8 *Suppose algorithm \mathcal{A} is k -piece piece-wise-constant. Then, for any $\epsilon > 0$, an estimated allocation rule $\mathbf{y}(\cdot)$ is k -piece piece-wise-constant, and is $\frac{\epsilon}{2}$ -close to $\mathbf{x}(\cdot)$ with probability at least $1 - \frac{\epsilon}{2}$. The construction of $\mathbf{y}(\cdot)$ requires $O(nk(\epsilon)^{-2} \log(2k/n\epsilon))$ black-box calls to \mathcal{A} .*

Proof: The execution time and the fact that $\mathbf{y}(\cdot)$ is k -piece piecewise constant follow immediately from the definition. Choose some i and let I_j denote piece j of the valuation space for agent i in \mathcal{A} , and write $x_i(I_j)$ for the (constant) value of $x_i(v)$ for any $v \in I_j$. By the Hoeffding-Chernoff inequality, the probability that $|y_{ij} - x_i(I_j)| > \epsilon/2$ is at most $e^{-4(\epsilon)^{-2} \log(2k/n\epsilon)(\epsilon/2)^2} \leq \epsilon/2kn$. Thus, taking the union bound over all i and j , we conclude that

$$|y_{ij} - x_i(I_j)| \leq \frac{\epsilon}{2}$$

for all i and j with probability at least $1 - \frac{\epsilon}{2}$. □

4.2.4 Statistical Ironing

We are now ready to combine our sampling procedure with the ironing procedure from the ideal model to construct an ϵ -BIC algorithm from a piecewise constant algorithm \mathcal{A} .

Definition 4.6 ($\tilde{\mathcal{A}}_\epsilon$) *Given piecewise constant algorithm \mathcal{A} and $\epsilon > 0$, the statistically ironed algorithm $\tilde{\mathcal{A}}_\epsilon$ for \mathcal{A} with error ϵ is:*

1. *Find the estimated allocation rule $\mathbf{y}(\cdot)$ for \mathcal{A} .*
2. *Find monotonizing intervals \mathcal{I} for $\mathbf{y}(\cdot)$ (as described in Section 3.1).*
3. *Run $\bar{\mathcal{A}}_\mathcal{I}(\mathbf{v})$.*

Lemma 4.9 *$\tilde{\mathcal{A}}_\epsilon$ is $2\epsilon v_{\max}$ -BIC and $\tilde{\mathcal{A}}_\epsilon \geq \mathcal{A} - n\epsilon\mu_{\max}$.*

Proof: By Lemma 4.8, $y_i(\cdot)$ is k -piece piece-wise-constant for each i . Let $\mathcal{A}_{\mathbf{y}}$ be the (fictional) algorithm with allocation rule \mathbf{y} . Since \mathcal{I} is the monotonizing interval set for $\mathcal{A}_{\mathbf{y}}$, if $\mathcal{A}_{\mathbf{y}}$ were ironed according to \mathcal{I} , the result would be $\bar{\mathcal{A}}_{\mathbf{y}}$ which is monotone.

By Lemma 4.8, $\mathcal{A}_{\mathbf{y}}$ is $\frac{\epsilon}{2}$ -close to \mathcal{A} with probability $1 - \frac{\epsilon}{2}$. In this case, Lemma 4.6 implies $\bar{\mathcal{A}}_{\mathcal{I}}$ is $\frac{\epsilon}{2}$ -close to $\bar{\mathcal{A}}_{\mathbf{y}}$. For the remaining probability, $\frac{\epsilon}{2}$, we note that $\bar{\mathcal{A}}_{\mathcal{I}}$ is trivially 1-close to $\bar{\mathcal{A}}_{\mathbf{y}}$. Thus, taking expectation over all possible outcomes of the sampling, we conclude that $\tilde{\mathcal{A}}_{\epsilon}$ is ϵ -close to monotone, and is therefore $2\epsilon v_{\max}$ -BIC by Lemma 4.4.

Since $\mathcal{A}_{\mathcal{I}}$ is $\frac{\epsilon}{2}$ close to $\bar{\mathcal{A}}_{\mathbf{y}}$ with probability $1 - \frac{\epsilon}{2}$, Lemma 4.5 and Lemma 4.7 imply that with probability $1 - \frac{\epsilon}{2}$,

$$\bar{\mathcal{A}}_{\mathcal{I}} \geq \bar{\mathcal{A}}_{\mathbf{y}} - \frac{1}{2}n\epsilon\mu_{\max} \geq \mathcal{A}_{\mathbf{y}} - \frac{1}{2}n\epsilon\mu_{\max} \geq \mathcal{A} - n\epsilon\mu_{\max}.$$

For the remaining probability, $\frac{\epsilon}{2}$, we note that trivially $\bar{\mathcal{A}}_{\mathcal{I}} \geq \mathcal{A} - \mathcal{A} \geq \mathcal{A} - n\mu_{\max}$. Thus, taking expectation over all possible outcomes of sampling, we conclude $\tilde{\mathcal{A}}_{\epsilon} \geq \mathcal{A} - n\epsilon\mu_{\max}$. \square

Theorem 4.2 now follows by taking \mathcal{A}' to be the statistically ironed algorithm $\tilde{\mathcal{A}}_{\epsilon'}$ applied to the discretization $\dot{\mathcal{A}}_{\epsilon'}$ of \mathcal{A} , where $\epsilon' = \epsilon/3n$. Then by Lemmas 4.9 and 4.7, \mathcal{A}' is $2\epsilon'v_{\max}$ -BIC, and hence ϵv_{\max} -BIC, and $\mathcal{A}' \geq \dot{\mathcal{A}}_{\epsilon'} - n\epsilon'\mu_{\max} \geq \mathcal{A} - 3n\epsilon'\mu_{\max} = \mathcal{A} - \epsilon\mu_{\max}$. The runtime (which is dominated by sampling in the construction of \mathbf{y}) is $O(nk\epsilon'^{-2} \log(2kn/\epsilon')) = \tilde{O}(n^3\epsilon^{-3} \log(v_{\max}/\epsilon\mu_{\max}))$, where recall $k = \log_{1+\epsilon}(v_{\max}/\epsilon\mu_{\max})$ is the number of discrete intervals in $\dot{\mathcal{A}}_{\epsilon'}$.

4.3 Bayesian Incentive-Compatibility

We now outline machinery by which we can make an ϵ -BIC mechanism BIC, outright, without much additional loss. This approach will apply to any ϵ -BIC mechanism with allocation rules that are piecewise constant.

4.3.1 An Explicitly Monotone Algorithm

Our approach will be to form a convex combination of an algorithm \mathcal{A} that is close to monotone with the following explicitly monotone algorithm.

Definition 4.7 Suppose \mathcal{A} has k -piece piece-wise constant allocation rules, and suppose S_1, \dots, S_n are allocations such that $i \in S_i$ for all i . The stair algorithm for \mathcal{A} , $\mathcal{S}^{\mathcal{A}}$, does the following:

1. Pick an agent i uniformly from the n agents.
2. If v_i is in the j th highest piece of k pieces, allocate to S_i with probability $(j-1)/(k-1)$.

We observe that if \mathcal{A} is ϵ -close to a monotone algorithm, then a convex combination of \mathcal{A} and $\mathcal{S}^{\mathcal{A}}$ will be monotone.

Lemma 4.10 If \mathcal{A}' with k -piece piece-wise-constant allocation rules is ϵ -close to a monotone \mathcal{A} , then $\hat{\mathcal{A}}$, the convex combination of \mathcal{A}' with probability $1 - \delta$ with $\mathcal{S}^{\mathcal{A}'}$ with probability δ , is BIC, where $\delta = 2(k-1)n\epsilon$.

Proof: To show $\hat{\mathcal{A}}$ is BIC, choose any agent i and any values $v_i < v_i'$; we will show $\hat{x}_i(v_i) \leq \hat{x}_i(v_i')$. If v_i, v_i' are in the same piece of the valuation space then $\hat{x}_i(v_i) = \hat{x}_i(v_i')$. Otherwise, since \mathcal{A}' is

ϵ -close to monotone \mathcal{A} , it must be that $x_i'(v_i) \leq x_i'(v_i') - 2\epsilon$. Furthermore, if $\mathbf{s}(\cdot)$ is the allocation rule for $\mathcal{S}^{\mathcal{A}'}$, then $s_i(v_i) \leq s_i(v_i') + 1/(k-1)n$. We conclude that

$$\begin{aligned}\hat{x}_i(v_i) &= (1-\delta)x_i'(v_i) + \delta s_i(v_i) \\ &\leq \hat{x}_i(v_i') - 2\epsilon + \delta/(k-1)n \\ &= \hat{x}_i(v_i')\end{aligned}$$

as required, since $\delta = 2(k-1)n\epsilon'$. \square

We like to apply Lemma 4.10 to $\tilde{\mathcal{A}}_\epsilon$ from Lemma 4.9, thereby proving Theorem 4.1. However, one issue has not been addressed: how to find sets S_1, \dots, S_n . In many settings finding such sets is trivial. For example, in downward-closed feasibility problems (e.g. combinatorial auctions), we could simply take $S_i = \{i\}$ and Theorem 4.1 follows immediately. However, in general settings, it may be difficult to find feasible sets (or low-cost sets in general cost settings). The remainder of this section will be devoted to describing a general process by which sets S_1, \dots, S_n can be found.

4.3.2 Implementing the Stair Algorithm in General Cost Settings

In general cost settings, algorithm $\mathcal{S}^{\mathcal{A}}$ may incur negative value if, for some i , the cost of set S_i is large relative to v_i . To bound the value of $\hat{\mathcal{A}}$ we must therefore limit the costs of sets S_1, \dots, S_n . Our upper bound on cost will depend on the following quantity, which relates to the structure of the piecewise constant intervals for algorithm \mathcal{A} .

Definition 4.8 Suppose \mathcal{A} has piece-wise constant allocation rules, where $\mathcal{I}_i = \{I_1, I_2, \dots\}$ are the constant intervals for agent i . The stair threshold for agent i , $w_i^{\mathcal{A}}$, is defined as $w_i^{\mathcal{A}} := \max I_1$. That is, $w_i^{\mathcal{A}}$ is the upper endpoint of the first valuation space interval for agent i .

We can now relate the value of $\hat{\mathcal{A}}$ to the cost of sets S_1, \dots, S_n and the stair thresholds of algorithm \mathcal{A} .

Lemma 4.11 If there exists $X \geq 0$ such that $c(S_i) \leq w_i^{\mathcal{A}} + X$ for all i , then $\hat{\mathcal{A}} \geq \mathcal{A} - \delta(n\mu_{\max} + X)$.

Proof: By construction, $\hat{\mathcal{A}} = (1-\delta)\mathcal{A} + \delta\mathcal{S}^{\mathcal{A}}$. Recall that $\mathcal{S}^{\mathcal{A}}$ chooses some i uniformly at random, and then either allocates S_i or \emptyset . Moreover, $\mathcal{S}^{\mathcal{A}}$ will always allocate \emptyset if v_i is in the first piece of the valuation space; that is, if $v_i < w_i^{\mathcal{A}}$. We therefore conclude that $\mathcal{S}^{\mathcal{A}} \geq \frac{1}{n} \sum_i \min\{w_i^{\mathcal{A}} - c(S_i), 0\} \geq -X$. Also, $\mathcal{A} \leq n\mu_{\max}$ trivially. Thus $\hat{\mathcal{A}} \geq \mathcal{A} - \delta n\mu_{\max} + \delta(-X) = \mathcal{A} - \delta(n\mu_{\max} + X)$. \square

Our goal will be to find sets S_i with $c(S_i) \leq w_i^{\mathcal{A}} + n\mu_{\max}/\sqrt{\epsilon}$, then apply Lemma 4.11 with $X = n\mu_{\max}/\sqrt{\epsilon}$. To find such sets, we will apply the same sampling techniques used in the construction of $\tilde{\mathcal{A}}_\epsilon$. That is, for each i and each piece of the valuation space, we will run \mathcal{A} on many sample inputs. As long as $x_i(I)$ is not too small on a given interval I , we are very likely to find some valid allocation that includes agent i during the sampling process! Moreover, we will show that not all sets discovered in this way can have high cost, so with high probability we will find a set S_i with cost at most $n\mu_{\max}/\sqrt{\epsilon}$. To relate the cost of set S_i with $w_i^{\mathcal{A}}$, we will also *iron together all left-most intervals for which a low-cost set was not found*. These ironed-together intervals will then act like a single piece of valuation space, which will allow us to relate the cost of any set S_i we do find to the stair threshold for the (modified) algorithm.

Definition 4.9 (\mathcal{A}_ϵ^s) Given piece-wise constant algorithm \mathcal{A} , the stair-compatible algorithm for \mathcal{A} , \mathcal{A}_ϵ^s , is as follows:

1. For each agent i :
2. Let $\mathcal{I}_i = \{I_1, \dots, I_k\}$ be the constant valuation space intervals for agent i .
3. For each $I_j \in \mathcal{I}_i$, draw $\epsilon^{-2} \log(n/2\epsilon)$ samples from \mathbf{F} conditional on $v_i \in I_j$, and run \mathcal{A} on each of these samples.
4. Let j_i be the minimal index such that, for some sample of interval I_{j_i} , \mathcal{A} allocated a set T_i with $T_i \ni i$ and $c(T_i) \leq \min I_{j_i} + n\mu_{\max}/\sqrt{\epsilon}$. Choose S_i to be any such T_i .
5. If no such set T_i was returned for any interval, take $j_i = k + 1$ and $S_i = \{i\}$.
6. Define $\mathcal{I}'_i = \{I_1 \cup \dots \cup I_{j_i-1}, I_{j_i}, \dots, I_k\}$.
7. Run $\bar{\mathcal{A}}_{\mathcal{I}'}(\mathbf{v})$.

Note that, as part of the execution of \mathcal{A}_ϵ^s , a set $S_i \ni i$ will be found for each i , which is taken to be any set satisfying the conditions on line 4 for interval I_{j_i} (or $\{i\}$ if no sets were found).

In summary, \mathcal{A}_ϵ^s samples each constant interval for agent i , searching for an appropriate set S_i . We take I_{j_i} to be the leftmost interval for which such a set S_i was found. All intervals to the left of I_{j_i} are then ironed together. Thus, regardless of the sampling outcome, I_{j_i} will be the second valuation space piece for agent i in algorithm $\bar{\mathcal{A}}_{\mathcal{I}'}$. Thus $c(S_i) \leq w_i^{\mathcal{A}_\epsilon^s} + n\mu_{\max}/\sqrt{\epsilon}$.

Lemma 4.12 *The stair compatible algorithm \mathcal{A}_ϵ^s for \mathcal{A} (Definition 4.9) and stair thresholds $\mathbf{w}^{\mathcal{A}_\epsilon^s}$ (Definition 4.8) satisfy $c(S_i) \leq w_i^{\mathcal{A}_\epsilon^s} + n\mu_{\max}/\sqrt{\epsilon}$ for all i .*

Proof: For each i , if no set satisfying the conditions on line 4 of \mathcal{A}_ϵ^s was found during the sampling of any interval, then $j_i = k+1$ and all intervals of \mathcal{A} are ironed together in \mathcal{I}' . In this case $w_i^{\mathcal{A}_\epsilon^s} = \infty$, so $c(S_i) \leq w_i^{\mathcal{A}_\epsilon^s}$ trivially. Otherwise, by line 4 of \mathcal{A}_ϵ^s , $c(S_i) \leq \min I_{j_i} + n\mu_{\max}/\sqrt{\epsilon}$. However, since all intervals to the left of I_{j_i} are ironed together in \mathcal{I}' , I_{j_i} will be the second piecewise constant interval of \mathcal{A}_ϵ^s , and hence $\min I_{j_i} = w_i^{\mathcal{A}_\epsilon^s}$. Thus $c(S_i) \leq \min w_i^{\mathcal{A}_\epsilon^s} + n\mu_{\max}/\sqrt{\epsilon}$ as required. \square

Lemma 4.13 *The stair compatible algorithm \mathcal{A}_ϵ^s for \mathcal{A} (Definition 4.9) satisfies $\mathcal{A}_\epsilon^s \geq \mathcal{A} - 2n\mu_{\max}/\sqrt{\epsilon}$.*

Proof: We claim that, with probability at least $1 - \frac{\epsilon}{2}$, for each agent i , the allocation rules for $\bar{\mathcal{A}}_{\mathcal{I}'}$ (from line 7 of \mathcal{A}_ϵ^s) and \mathcal{A} will differ only on values v_i for which $x_i(v_i) \leq \sqrt{\epsilon} + \frac{\epsilon}{2}$. Before proving the claim, let us see how it implies the desired result. The claim implies that $\bar{\mathcal{A}}_{\mathcal{I}'} \geq \mathcal{A} - (\sqrt{\epsilon} + \frac{\epsilon}{2})n\mu_{\max}$ with probability $1 - \frac{\epsilon}{2}$. For the remaining probability, we note that $\bar{\mathcal{A}}_{\mathcal{I}'} \geq \mathcal{A} - \mathcal{A} \geq \mathcal{A} - n\mu_{\max}$ trivially. Thus, over all possible outcomes of sampling, we conclude that

$$\mathcal{A}_\epsilon^s \geq \mathcal{A} - \left(\sqrt{\epsilon} + \frac{\epsilon}{2}\right)n\mu_{\max} - \frac{\epsilon}{2}n\mu_{\max} = \mathcal{A} - \sqrt{\epsilon}n\mu_{\max}$$

as required.

Let us now prove the claim. Choose some agent i and suppose that $\bar{\mathcal{A}}_{\mathcal{I}'}$ and \mathcal{A} differ on some interval I with $x_i(I) \geq \sqrt{\epsilon} + \frac{\epsilon}{2}$. Let I be the leftmost such interval. For the remainder of the proof we will say that a set T has *low cost for I* if $c(T) \leq \min I + n\mu_{\max}/\sqrt{\epsilon}$. Then, by the definition of \mathcal{I}'_i , it must be that no set $T \ni i$ with low cost was found during the sampling of interval I

for agent i . Let us bound the probability of this event. Given $\mathbf{v} \sim \mathbf{F}$, let $B(\mathbf{v})$ be the event $[x_i(\mathbf{v}) \wedge \sum_i v_i \leq \min I + n\mu_{\max}/\sqrt{\epsilon}]$. If event $B(\mathbf{v})$ occurs for some sample \mathbf{v} , this means that \mathcal{A} returned some allocation $T \ni i$ and furthermore $\sum_i v_i \leq \min I + n\mu_{\max}/\sqrt{\epsilon}$. But note that this allocation must generate non-negative profit (otherwise it would never be allocated), and hence T must have low cost for I . Thus $B(\mathbf{v})$ is precisely the event that \mathcal{A} returns a set $T \ni i$ with low cost for I .

Consider the probability of $B(\mathbf{v})$. By Markov's inequality, $\Pr_{\mathbf{v}}[\sum_{j \neq i} v_j > n\mu_{\max}/\sqrt{\epsilon}] < \sqrt{\epsilon}$. Thus, since $v_i \geq \min I$ with probability 1 conditional on $v_i \in I$, $\Pr_{\mathbf{v}}[\sum_i v_i > \min I + n\mu_{\max}/\sqrt{\epsilon}] < \sqrt{\epsilon}$. Also, $\Pr_{\mathbf{v}}[\neg x_i(\mathbf{v}) \mid v_i \in I] = 1 - x_i(I) \leq 1 - (\sqrt{\epsilon} + \frac{\epsilon}{2})$. The union bound then implies that $\Pr_{\mathbf{v}}[\neg B(\mathbf{v})] \leq 1 - (\sqrt{\epsilon} + \frac{\epsilon}{2}) + \sqrt{\epsilon} = 1 - \frac{\epsilon}{2}$, so $\Pr_{\mathbf{v}}[B(\mathbf{v})] \geq \frac{\epsilon}{2}$.

By Chernoff-Hoeffding inequality, the probability that event B does not occur even once during $\epsilon^{-2} \log(2n/\epsilon)$ samples is at most $\frac{\epsilon}{2n}$. We conclude that the probability that no set $T \ni i$ with low cost was found during the sampling of interval I is at most $\frac{\epsilon}{2n}$. This is therefore a bound on the probability that $\hat{\mathcal{A}}_I$ and \mathcal{A} differ for agent i on some interval I with $x_i(I) \geq \sqrt{\epsilon} + \frac{\epsilon}{2}$. By the union bound, the probability that this occurs for *any* agent is at most $\frac{\epsilon}{2}$, as required. \square

We are now ready to describe algorithm $\hat{\mathcal{A}}_\epsilon$ from the statement of Theorem 4.1.

Definition 4.10 ($\hat{\mathcal{A}}_\epsilon$) *Given algorithm \mathcal{A} and $\epsilon > 0$, the monotoneization of \mathcal{A} , $\hat{\mathcal{A}}_\epsilon$, proceeds as follows:*

1. Construct $\dot{\mathcal{A}}_\epsilon$, the discretized version of \mathcal{A} .
2. Construct \mathcal{A}_ϵ^s , the stair-compatible version of $\dot{\mathcal{A}}_\epsilon$. This generates sets S_1, \dots, S_n .
3. Construct $\tilde{\mathcal{A}}_\epsilon$, the statistically ironed algorithm for \mathcal{A}_ϵ^s .
4. With probability $\delta = 2k(n-1)\epsilon$, execute $\mathcal{S}^{\mathcal{A}_\epsilon^s}$ with sets S_1, \dots, S_n . Else, execute $\tilde{\mathcal{A}}_\epsilon$.

Lemma 4.14 $\hat{\mathcal{A}}_\epsilon$ is BIC, and $\hat{\mathcal{A}}_\epsilon \geq \mathcal{A} - 7kn^2\sqrt{\epsilon}\mu_{\max}$.

Proof: Lemma 4.9 implies that $\tilde{\mathcal{A}}_\epsilon$ is ϵ -close to a monotone algorithm, and during the construction of \mathcal{A}_ϵ^s we find sets S_1, \dots, S_n with $S_i \ni i$. Thus the convex combination of $\tilde{\mathcal{A}}_\epsilon$ with $\mathcal{S}^{\mathcal{A}_\epsilon^s}$ is BIC by Lemma 4.10.

We note that costs are not affected by our ironing techniques, and, by Lemma 4.13,

$$\mathcal{A}_\epsilon^s \geq \dot{\mathcal{A}}_\epsilon - 2\sqrt{\epsilon}n\mu_{\max} \geq \mathcal{A} - 3\sqrt{\epsilon}n\mu_{\max}.$$

Also, $c(S_i) \leq w_i^{\mathcal{A}_\epsilon^s} + n\mu_{\max}/\sqrt{\epsilon}$ for all i by Lemma 4.12. Thus, by Lemma 4.11,

$$\begin{aligned} \hat{\mathcal{A}}_\epsilon &\geq \mathcal{A}_\epsilon^s - \delta(n\mu_{\max} + n\mu_{\max}/\sqrt{\epsilon}) \\ &\geq \mathcal{A} - 3n\sqrt{\epsilon}\mu_{\max} - 2(2(k-1)n\epsilon)n\mu_{\max}/\sqrt{\epsilon} \\ &\geq \mathcal{A} - 7kn^2\sqrt{\epsilon}\mu_{\max}. \end{aligned}$$

\square

Theorem 4.1 now follows immediately from Lemma 4.14 by considering algorithm $\hat{\mathcal{A}}_{\epsilon'}$, where $\epsilon' = (\epsilon/7kn^2)^2 = \epsilon^2/49k^2n^4$. The runtime, which is dominated by the sampling in the construction of $\hat{\mathcal{A}}_\epsilon$, is $O(kn(\epsilon')^{-2}) = \tilde{O}(n^9\epsilon^{-9} \log^5(v_{\max}/\epsilon\mu_{\max}))$.

4.3.3 Feasibility Settings

In general feasibility settings, where costs are either 0 or infinite, the performance of algorithm $\hat{\mathcal{A}}_\epsilon$ improves significantly. Specifically, we can improve Lemma 4.13 as follows:

Lemma 4.15 *In feasibility settings, $\mathcal{A}_\epsilon^s \geq \mathcal{A} - 2\epsilon n \mu_{\max}$.*

Proof: Consider some agent i and interval $I \in \mathcal{I}$, and suppose $x_i(I) \geq \frac{\epsilon}{2}$. Consider the probability of finding an allocation with cost at most $\min I + \epsilon^{-1/2} n \mu_{\max}$ when sampling for this interval. Since costs are either 0 or ∞ , this is precisely the probability of finding an allocation that includes agent i , which is $x_i(I) \geq \frac{\epsilon}{2}$. By Chernoff-Hoeffding inequality, the probability that this event does not occur even once in $\epsilon^{-2} \log(n/2\epsilon)$ samples is at most $\epsilon/2n$. We will therefore successfully find a set $S_i \ni i$ with probability at least $1 - \epsilon/2n$.

For each i , let I_{j_i} denote the leftmost interval on which $x_i(I) \geq \epsilon$. By the union bound, with probability $1 - \epsilon/2$ we will find a set $S_i \ni i$ when sampling interval I_{j_i} , for all i . In this case, the behavior of algorithms \mathcal{A}_ϵ^s and \mathcal{A} differ only on intervals I to the left of I_{j_i} , all of which satisfy $x_i(I) < \frac{\epsilon}{2}$. Thus, conditioning on an event of probability $1 - \frac{\epsilon}{2}$, $\mathcal{A}_\epsilon^s \geq \mathcal{A}(1 - \frac{\epsilon}{2}) \geq \mathcal{A} - n \mu_{\max} \epsilon$. For the remaining probability, $\frac{\epsilon}{2}$, we note that $\mathcal{A} \leq n \mu_{\max}$ trivially. We conclude that $\mathcal{A}_\epsilon \geq \mathcal{A} - 2\epsilon n \mu_{\max}$ unconditionally. \square

Using Lemma 4.15 instead of Lemma 4.13 in the analysis of $\hat{\mathcal{A}}_\epsilon$, we find that the statement of Lemma 4.14 improves to show that $\hat{\mathcal{A}}_\epsilon > \mathcal{A} - 7kn^2 \epsilon \mu_{\max}$ in feasibility settings. The second half of Theorem 4.1 then follows by considering $\hat{\mathcal{A}}_{\epsilon'}$ with $\epsilon' = 7kn^2 \epsilon$, which has a runtime of $O(kn(\epsilon')^{-2}) = \tilde{O}(n^5 \epsilon^{-5} \log^3(v_{\max}/\epsilon \mu_{\max}))$.

5 Conclusions and Open Questions

Our main result is for the setting of single-parameter agents and the objective of social welfare where we give a black-box reduction that converts any Bayesian approximation algorithm into a Bayesian incentive compatible mechanism. For these settings there is no gap separating the approximation complexity of algorithms and BIC mechanisms. While this result is extremely general, the situations not covered by our main theorem are of notable interest.

1. Multi-parameter Bayesian mechanism design is not very well understood; however, there is every reason to believe that approximation (which has not been pursued much by the economics literature) has a very interesting and relevant role to play in providing positive results. A notable example is in approximations for Bayesian auctions for unit-demand agents (for the objective of profit maximization). Here the performance of an optimal single-parameter mechanism gives an upper bound on that of the optimal multi-parameter mechanism, and a mechanism for the multi-parameter setting is able to simulate the outcome of this optimal single-parameter mechanism without much loss in performance. See [6] for details.

We would like to know whether there is a gap separating the approximation complexity of algorithms and BIC mechanisms in multi-parameter settings for social welfare maximization. This question is set in the context of a recent lower bound by Papadimitriou et al. [13] for the combinatorial public project problem that shows that there is a gap separating the approximation complexity of algorithms and ex post IC mechanisms. This result, however, does not immediately extend to show a separation for BIC mechanisms. An important first

step in addressing the Bayesian multi-parameter setting is captured by the following question. *Given a black-box for a weakly-monotone allocation rule,⁶ can can BIC payments be computed in polynomial time?* (That is, can one generalize Archer et al.’s payment computation from the single-parameter setting [2] to the multi-parameter setting?)

2. Our reduction applies to the objective of social welfare maximization. It would be nice to extend our result more generally to any monotone objective function, e.g. makespan. Unfortunately, our approach fails to preserve the approximation factor of the makespan objective. For welfare our ironing procedure (weakly) increases the objective because we become more likely to allocate to a higher-valued agent at the expense of being less likely to allocate to the same agent with a lower value. For makespan, allocating in the higher-valued case may not improve the objective at all if the makespan is given by another agent. Thus, our loss for allocating less often in the lower-valued case is not offset by any improvement. A concrete example of this is given in Appendix A.1. *Is there a polynomial-time reduction that turns any approximation algorithm for any monotone objective into a BIC mechanism with the same approximation factor?*
3. In the special case where our reduction is applied to a worst-case c -approximation (recall: our reduction applies more generally to Bayesian c -approximations), the resulting BIC mechanism is still only an c -approximation in the weaker Bayesian sense. A concrete example of this is given in Appendix A.2. *Is there a polynomial-time black-box reduction that turns any worst-case c -approximation algorithm into a BIC mechanism that is also a worst-case c -approximation?*
4. While Bayes-Nash equilibrium (i.e., BIC) is the standard equilibrium concept for implementation in economics, the stronger dominant strategy equilibrium (i.e., ex post IC) is the standard equilibrium concept for implementation in computer science. The main challenge in obtaining a similar reduction for IC mechanisms is that the valuation space is exponentially big and monotonizing all points seems to require an exhaustive procedure. The key to the approach of our BIC reduction is that resampling bids from the distribution can be done independently for each agent without changing an agent’s prior on the other agent’s bids. This does not seem to be the case in non-Bayesian settings. One potential approach would be to apply our ironing technique repeatedly, re-ironing an agent’s curve whenever it is affected by an ironing of another agent’s curve. Such a procedure can be argued to terminate, but will not generally give a monotone allocation rule. A concrete example is given in Appendix A.4. *Is there a polynomial-time reduction for turning any $f(n)$ -approximation (worst-case or Bayesian) algorithm for a single-parameter domain into an ex post IC mechanism that is a (worst-case or Bayesian) $\Theta(f(n))$ -approximation?*

The final question above can be rephrased intuitively as our main open question. *Is there a gap separating the approximation complexity of implementation by Bayesian incentive compatible and ex post incentive compatible mechanisms?*

⁶Weak-monotonicity is a generalization of the monotone allocation rule requirement to multi-parameter settings.

References

- [1] K. Akcoglu, J. Aspens, B. Dasgupta, and M. Kao. An opportunity-cost algorithm for combinatorial auctions. In *Applied Optimization: Computational Methods in Decision-Making, Economics, and Finance*, 2002.
- [2] A. Archer, C. Papadimitriou, K. Talwar, and E. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. In *Proc. 14th ACM Symp. on Discrete Algorithms*. ACM/SIAM, 2003.
- [3] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, 2001.
- [4] M. Babaioff and L. Blumrosen. Computationally-feasible truthful auctions for convex bundles. In *Proc. 7th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2004.
- [5] M. Babaioff, R. Lavi, and E. Pavlov. Single-value combinatorial auctions and algorithmic implementation in undominated strategies. *Journal of the ACM*, 2009.
- [6] S. Chawla, J. Hartline, and R. Kleinberg. Algorithmic pricing via virtual valuations. In *Proc. 9th ACM Conf. on Electronic Commerce*, 2007.
- [7] S. Chawla, J. Hartline, U. Rajan, and R. Ravi. Bayesian optimal no-deficit mechanism design. In *Workshop on Internet and Network Economics (WINE)*, 2006.
- [8] G. Christodoulou, A. Kovács, and Michael Schapira. Bayesian combinatorial auctions. In *Proc. 35th Intl. Colloq. on Automata, Languages and Programming*, pages 820–832, 2008.
- [9] P. Dhangwatnotai, S. Dobzinski, S. Dughmi, and T. Roughgarden. Truthful approximation schemes for single-parameter agents. In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, 2008.
- [10] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.
- [11] D. Lehmann, L. I. O’Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. In *Proc. 1st ACM Conf. on Electronic Commerce*, pages 96–102. ACM Press, 1999.
- [12] R. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6:58–73, 1981.
- [13] C. Papadimitriou, M. Schapira, and Y. Singer. On the hardness of being truthful. In *Proc. 49th IEEE Symp. on Foundations of Computer Science*, 2008.

A Counterexamples

A.1 Beyond Social Welfare

In the ideal model, our general reduction applies to any single-parameter optimization problem and converts an algorithm into a mechanism with at least the same expected social welfare. Unfortu-

	10	1000	1001
100	1, 0	0, 1	0, 1
1	0, 1	1, 1	1, 1

Figure 3: The allocation rule for algorithm \mathcal{A} . The vertical axis corresponds to v_1 , the horizontal to v_2 , and the table entries are of the form “ x_1, x_2 ”. For example, if $(v_1, v_2) = (100, 10)$, then $(x_1, x_2) = (1, 0)$.

nately, this approach does not preserve other relevant objective values, even monotone ones such as the makespan. We illustrate this deficiency of the approach with an example.

Consider the problem of *job scheduling on related machines* with the goal of minimizing makespan. Here the machines are agents and each has a (privately-known) speed. The time a job takes on a machine is the product of its length and the machine’s speed. The goal of the mechanism is to assign a given set of jobs with varying lengths to the machines so as to minimize the time until all machines have finished processing their jobs, a.k.a., the makespan.

Consider an instance in which we have 10 unit-length jobs, and 5 machines. We assume a Bayesian setting, where the speeds of the machines are probabilistic. The first 4 machines are identical: they all have speed 2 with probability 1. The last machine has either speed 1 or speed 2, each with probability 1/2.

Suppose \mathcal{A} behaves in the following way. When $s_5 = 2$, it will choose $x_i = 2$ for all i , resulting in a makespan of 1. When $s_5 = 1$, \mathcal{A} sets $\mathbf{x} = (6, 1, 0, 0, 3)$, resulting in a makespan of 3 (whereas the optimal is 1.5). Thus the average expected makespan achieved by \mathcal{A} is 2.

We note that, for this algorithm, the allocation curve for machine 5 is not monotone. Our monotonization procedure will therefore iron the valuation space of machine 5. The optimal ironing of this curve will draw s'_5 uniformly from $\{1, 2\}$. This causes 4 equally likely possibilities, corresponding to $(s_5, s'_5) \in \{1, 2\}^2$.

If $s_5 = s'_5$ then \mathcal{A} is proceeding as though no ironing occurred: if $s_5 = s'_5 = 1$ then the makespan is 1, and if $s_5 = s'_5 = 2$ the makespan is 3. Suppose $s_5 = 2, s'_5 = 1$. Then \mathcal{A} forms allocation x as though machine 5 has speed 1, though it actually has speed 2. Hence we obtain $\mathbf{x} = (6, 1, 0, 0, 3)$, for a makespan of 3. If, on the other hand, $s_5 = 1, s'_5 = 2$ then we obtain $\mathbf{x} = (2, 2, 2, 2, 2)$, for a makespan of 2.

We conclude that the expected makespan of the ironed procedure is $\frac{1+3+3+2}{4} = 2.25$, which is strictly worse than the expected makespan obtained by the original algorithm.

A.2 Failure to Preserve Worst-Case Approximations

We present an example to demonstrate that ideal ironing does not preserve worst-case approximation ratios. Consider an auction of 2 objects to 2 unit-demand bidders, with the goal of optimizing social welfare. The private value of agent 1 is drawn uniformly from $\{1, 100\}$, and the private value of agent 2 is drawn uniformly from $\{10, 1000, 1001\}$. Let \mathcal{A} be an approximation algorithm whose allocation rule is described in Figure 3.

We note that \mathcal{A} is a worst-case 11/10 approximation algorithm (where the optimal solution is to always allocate to both players). Also, \mathcal{A} is not BIC for agent 1: $E[x_1(1)] = 2/3$, whereas $E[x_1(100)] = 1/3$. The ideal monotonization procedure will draw a new bid v'_1 for agent 1 uniformly from $\{1, 100\}$, and run \mathcal{A} on (v'_1, v_2) . Call this new algorithm $\bar{\mathcal{A}}$.

Note that if $v_1 = 100$ and $v_2 = 10$, then with probability $1/2$ $\bar{\mathcal{A}}$ will take $v'_1 = 1$ and choose allocation $(0, 1)$, and with the remaining probability it will take $v'_1 = 100$ and choose allocation $(1, 0)$. Hence, for this set of input values, the expected welfare obtained by $\bar{\mathcal{A}}$ is $\frac{100+10}{2} = 55$. Since 110 is optimal, $\bar{\mathcal{A}}$ is at best a 2-approximation algorithm, whereas \mathcal{A} is an $11/10$ -approximation algorithm. We conclude that ideal ironing can cause a significant decrease in worst-case approximation ratios.

A.3 Ironing Allocation Rules vs. Ironing Virtual Valuations

The ironing procedure described in this paper is reminiscent of the ironing procedure used by Myerson for maximizing revenue [12]. Myerson's optimal mechanism allocates to maximize virtual value (defined by $\phi(v) = v - \frac{1-F_i(v)}{f_i(v)}$). When the virtual valuation functions are non-monotone, this mechanism first irons the virtual valuation functions, then allocates the maximize ironed virtual value. A natural question is whether this is identical to ironing the non-monotone allocation rule that would result from maximizing non-ironed virtual values. We show that these two mechanisms are distinct; therefore the approach of ironing allocations is suboptimal in terms of revenue.

Consider the following distribution. With probability $1/2$, the value is drawn uniformly from interval $[10, 11]$. With the remaining probability, the value is drawn uniformly from $[11, 15]$. This results in a virtual valuation function defined by

$$\phi(v) = \begin{cases} 2v - 12 & v \in [10, 11] \\ 2v - 15 & v \in (11, 15]. \end{cases}$$

The motivation behind this distribution is that $\phi(v) > 0$ for all $v \in [10, 15]$, and the minimum of $\phi(v)$ does not occur at 10: $\phi(10) > \phi(11 + \epsilon)$ for sufficiently small ϵ .

Consider the allocation rule specified by assigning the item to the agent with the highest (non-ironed) virtual value. By symmetry, all players' allocation curves are the same, say $x(\cdot)$. Note $x(10) > 0$, since there is a non-zero probability that all other players have virtual values less than $\phi(10)$.

Next consider the allocation to maximize ironed virtual valuations. Let ϕ' denote the ironed virtual valuation function. Then ϕ' is monotone, and it can be verified that $\phi'(10) < \phi'(v)$ for all $v > 10$. Hence, if $x'(\cdot)$ is the allocation curve for this mechanism, we have $x'(10) = 0$.

Finally, consider the allocation rule $x''(\cdot)$ that would result from ironing the allocation rule $x(\cdot)$ as suggested by Definition 3.1. Since $x(10) > 0$ and $x(v) \geq 0$ for all $v \in [10, 15]$, it must be that $x''(10) > 0$ as it is a probability weighted average of some interval containing 10.

Thus, since $x'(10) = 0 \neq x''(10)$, we conclude that $x' \neq x''$ and hence these two different ironing procedures result in different mechanisms.

A.4 Recursive Ironing does not Guarantee ex post Incentive Compatibility

In order for a mechanism to guarantee ex-post incentive compatibility, it must be that, for all $i \in [n]$, the allocation rule x_i is monotone *for any choice of* \mathbf{v}_{-i} . Monotonizing each agent's allocation rule independently is insufficient to obtain this goal. Indeed, it is easy to construct examples where each agent's allocation curve is monotone in expectation, but non-monotone for a particular choice of the other agents' bids.

One might imagine the following recursive approach for obtaining ex-post incentive compatibility. Begin by assuming that each agents' input is drawn from the singleton interval $I_i = [v_i, v_i]$,

	1	2	3	4	5	6
2	0.20	0.60	0.60	0.20	0.20	0.60
1	0.80	0.20	0.82	0.22	0.84	0.24

Figure 4: The allocation rule for algorithm \mathcal{A} . The vertical axis corresponds to possible values v_1 , the horizontal axis corresponds to possible values v_2 , and the table entries denote the probability of allocation to both agents. For example, if $(v_1, v_2) = (1, 4)$, then $(x_1, x_2) = (0.22, 0.22)$.

	1	2	3	4	5	6
2	0.40	0.40	0.40	0.40	0.40	0.40
1	0.50	0.50	0.52	0.52	0.54	0.54

(a)

	1	2	3	4	5	6
2	0.45	0.45	0.46	0.46	0.47	0.47
1	0.45	0.45	0.46	0.46	0.47	0.47

(b)

Figure 5: The results of the recursive monotonization procedure for algorithm \mathcal{A} , on input $(1, 5)$, (a) after 1 step, (b) after 2 steps.

and let \mathbf{I} denote the cube $I_1 \times I_2 \times \cdots \times I_n$. Choose some agent i whose allocation curve is not monotone, under the assumption that each other agents' values are drawn from cube \mathbf{I} , and iron that agent's curve under this assumption. This ironing process may enlarge the interval from which agent i 's value is drawn; update I_i to be this new interval. Repeat this process, choosing a new agent on each iteration, until all agents' curves are monotone.

Unfortunately, as we now demonstrate, the above procedure fails to guarantee ex-post incentive compatibility. Consider an auction setting with 2 agents, such that either both agents receive an allocation or neither does. Suppose \mathcal{A} is the algorithm with allocation rule described in Figure 4.

Consider the application of our recursive monotonization technique on this algorithm when $(v_1, v_2) = (1, 5)$. Suppose we choose to monotone the allocation curve for agent 2. Applying our monotonization procedure to the values $(0.8, 0.2, 0.82, 0.22, 0.84, 0.24)$, we obtain ironed intervals $\{1, 2\}$, $\{3, 4\}$, and $\{5, 6\}$. The resulting allocation rule is shown in figure 5(a).

We next monotone curve x_1 at the point $v_2 = 5$. This curve is non-monotone for agent 1, and the resulting ironed interval is $\{1, 2\}$. The resulting allocation rule is shown in figure 5(b). After this monotonization, the allocation curves for all agents are monotone. The final expected allocation probability for both players is 0.47 (the entry at $(v_1, v_2) = (1, 5)$).

Next consider the application of this technique when $(v_1, v_2) = (2, 5)$. Monotonizing agent 2 first, we apply our procedure to values $(0.2, 0.6, 0.6, 0.2, 0.2, 0.6)$ and we obtained ironed interval $\{2, 3, 4, 5\}$. The resulting allocation rule is shown in figure 6(a).

We next monotone curve x_1 at the point $v_2 = 5$. This curve is non-monotone for agent 1, and the resulting ironed interval is $\{1, 2\}$. The resulting allocation rule is shown in figure 6(b). After this monotonization, the allocation curve for agent 2 is no longer monotone, so we must iron again over the interval $\{1, 2, 3, 4, 5, 6\}$. At this point all agents' allocation curves are monotone. The final expected allocation probability for both players is 0.46.

What we have shown is that, given $v_2 = 5$, our recursive monotonization procedure generates the allocation rule $x_1(1) = 0.47 > 0.46 = x_1(2)$ for agent 1. This procedure therefore does not result in a monotone allocation rule, and hence does not obtain ex-post incentive compatibility.

	1	2	3	4	5	6
2	0.20	0.40	0.40	0.40	0.40	0.60
1	0.80	0.52	0.52	0.52	0.52	0.24

(a)

	1	2	3	4	5	6
2	0.50	0.46	0.46	0.46	0.46	0.42
1	0.50	0.46	0.46	0.46	0.46	0.42

(b)

	1	2	3	4	5	6
2	0.46	0.46	0.46	0.46	0.46	0.46
1	0.46	0.46	0.46	0.46	0.46	0.46

(c)

Figure 6: The results of the recursive monotonization procedure for algorithm \mathcal{A} , on input $(2, 5)$, (a) after 1 step, (b) after 2 steps, (c) after 3 steps.